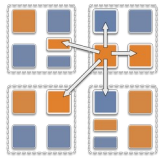
Chare Arrays

- Indexed collections of chares
  - Every item in the collection has a unique index and proxy
  - Can be indexed like an array or by an arbitrary object
  - Can be sparse or dense
  - Elements may be dynamically inserted and deleted

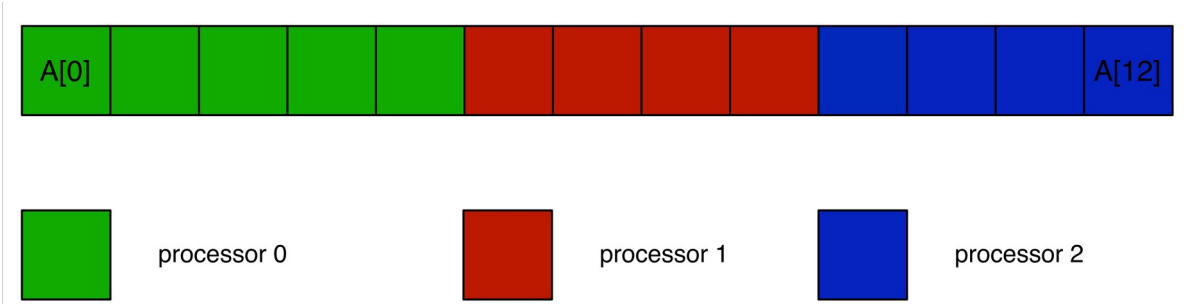
- For many scientific applications, collections of chares are a convenient abstraction
- Instead of creating networks of chares that learn about each other (by sending proxies to each other), each element in a chare array knows about all the others



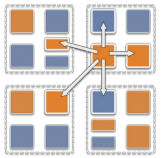
PPL UIUC

Chare Array Location

- By default, chare arrays are distributed across the available processors



- (This is one possible initial assignment for 1D chare arrays)
- Chare array elements can be migrated by the user or the runtime (load balancing)



PPL UIUC

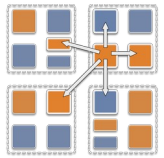
Declaring a Chare Array

.ci file:

```
array [1d] foo {
   entry foo(); // constructor
   // … entry methods …
}
array [2d] bar {
   entry bar(); // constructor
   // … entry methods …
}
```

.cpp file:

```
struct foo : public CBase_foo {
   foo() { }
   foo(CkMigrateMessage*) { }
   // … entry methods …
};
struct bar : public CBase_bar {
   bar() { }
   bar(CkMigrateMessage*) { }
   // … entry methods …
};
```



PPL UIUC

Constructing a Chare Array

- Constructed much like a regular chare
- The size of each dimension is passed to the constructor at the end

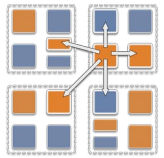
```
void someMethod() {
   CProxy_foo myFoo = CProxy_foo::ckNew(<params>, 10);   // 1d, size 10
   CProxy_bar myBar = CProxy_bar::ckNew(<params>, 5, 5); // 2d, size 5x5
}
```

- The proxy doesn't have to be retained:

```
CProxy_foo::ckNew(10);
```

- The proxy represents the entire array, and may be indexed to obtain a proxy to an individual element in the array

```
myFoo[4].invokeEntry(…);
myBar(2,4).method3(…);
```



PPL UIUC

thisIndex

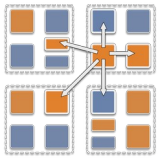
- 1d: `thisIndex` returns the index of the current chare array element
- 2d: `thisIndex.x` and `thisIndex.y` return the indices of the current chare array element

.ci file:

```
array [1d] foo
{
   entry foo();
}
```

.cpp file:

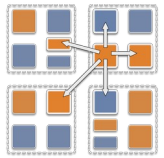
```
struct foo : public CBase_foo {
   foo() {
      CkPrintf("array index = %d",
thisIndex);
   }
};
```



PPL UIUC

Chare Array: Hello Example

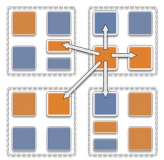
```
mainmodule arr {
    mainchare Main {
        entry Main(CkArgMsg*);
    }
    array [1D] hello {
        entry hello(int);
        entry void printHello();
    }
}
```



PPL UIUC

Chare Array: Hello Example

```
#include "arr.decl.h"
struct Main : CBase_Main {
   Main(CkArgMsg* msg) {
      int arraySize = atoi(msg->argv[1]);
      CProxy_hello p = CProxy_hello::ckNew(arraySize,
arraySize);
      p[0].printHello();
   }
};
struct hello : CBase_hello {
   int arraySize;
   hello(int n) : arraySize(n) { }
   void printHello() {
      CkPrintf("PE[%d]: hello from p[%d]\n", CkMyPe(),
thisIndex);
      if (thisIndex == arraySize – 1) CkExit();
      else thisProxy[thisIndex + 1].printHello();
   }
};
```



PPL UIUC